Mars Hab-Bot: Using MDPs to simulate a robot constructing human-livable habitats on Mars

Abraham Botros
SUNet ID: abotros
abotros@stanford.edu
Stanford University, Fall 2015 - AA228 Final Project Report

Abstract – As mankind begins planning future manned missions to Mars, it is becoming increasingly pressing for researchers and mission planners to design habitats that can be built (likely with the help of robots) to house human astronauts when they arrive on the Red Planet. In light of this, we formulate a Markov Decision Process (MDP) that simulates a simplistic setting in which a robot must place several habitat structure components at designated locations in a specific order. Results show this formulation proves effective for solving for optimal policies that allow the robot to complete all tasks as desired with minimal errors even under relatively high uncertainty in intended versus expected outcome of any given action. This provides a very basic but effective base for building more complex and realistic simulations of a habitat-building robot on Mars.

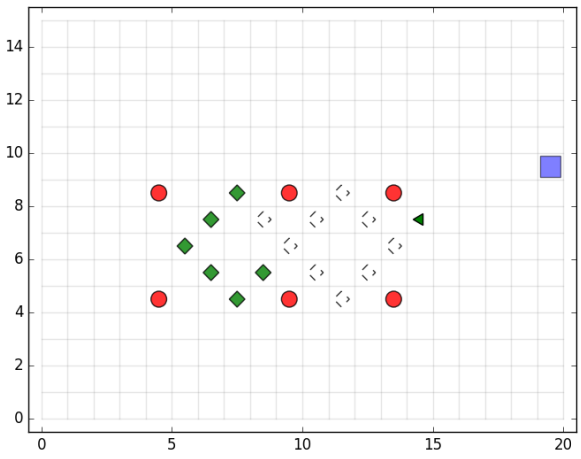


Figure 1: Example of visualization of robot placing habitat structure components in one of the simulated worlds.

1 Introduction

Humanity's future may lie out in the great expanse of space. Indeed, in the near future, the first humans may step foot on - and even colonize - other worlds beyond our pale blue dot. From NASA's "Journey to Mars" [8] to put boots on the Red Planet in the next couple of decades, to SpaceX's Elon Musk seeking to not only visit but colonize Mars in high numbers [4], both governmental and private efforts are underway to make Mars a new home-away-from-home for mankind.

However, for humans to live in such an extreme and unforgiving environment with limited resources, especially for flesh-and-blood organisms, we will likely require robots to establish human-livable habitats on the Martian surface before any substantial number of human colonists can have a sustained presence on Mars. Concerted effort is already being taken in experimenting with, designing, and planning such habitats, along with the logistics and risks associated with humans living in such habitats over long periods of time [6,7]. In addition, due to the complexities, (feedback) delays, and a variety of other issues regarding direct communication between a rover or robot on the Mars surface to ground control on Earth at NASA, manually steering, guiding, and instructing robots on Mars on how to exactly navigate and perform certain tasks is infeasible [2, 3]. As a result, any habitat-building robots will need to function largely autonomously, after given high-level instructions beforehand (such as ideal habitat specifications, locations, etc.).

In this project, we model a (very) simplified version of such an autonomous robot (dubbed a Mars "Hab-Bot"), its navigation, and its tasks as an MDP. The robot navigates a simulated, generated environment, with specific rewardable tasks to complete (placement of habitat structure components) in a pre-ordained order. The robot transitions probabilistically between its states, with uncertainty in the results of its intended movements and actions. Thus, the robot must deal with uncertainty at each step in its process. Value iteration is used to solve for an optimal policy under uncertainty, allowing the robot to plan for optimal moves no matter where it ends up in its world. Performance is evaluated on three different worlds, with different base locations, target locations, and probability parameters in each world. The results are quite promising, with policies performing optimally and maximizing rewards as hoped for, even in the face of high uncertainty.

2 Previous work

2.1 Relevant literature

Richard Bellman is credited with much of the pioneering work in sequential decision problems around the 1950's [11], including the establishment of MDPs as a meaningful and effective way for approaching these settings. As also summarized in [14], MDPs involve a rational agent that chooses action a_t at time t based on observing state s_t; the agent then receives a reward r_t and enters state s_{t+1}. This continues over time as an agent seeks to accrue maximal rewards by taking op-

timal actions according to some optimal policy, π^*. To plan appropriately to maximize rewards, the agent relies on two functions:

- A state transition function, $T(s' \mid s, a)$, which represents the probability of transitioning from state s to s' after executing action a.
- A reward function, $R(s, a)$, which represents the expected reward received when executing action a from state s.

As in [11] and Section 4.2.4 of [14], we can solve for optimal policies using the value iteration algorithm. This algorithm is based on the Bellman Equation:

$$U^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} T(s' \mid s, a) U^*(s') \right) \quad (1)$$

This can be shown to provide the value function for the optimal policy in the case of an infinite horizon problem with discount γ. Using this, value iteration iteratively updates estimates of the value function U until it converges to the optimal value function U^*, from which we can extract an optimal policy $\pi^*(s)$ using:

$$\pi^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s'} T(s' \mid s, a) U^*(s') \right) \quad (2)$$

Countless research works have been done making use of these formulations of MDPs and value iteration. One of the simplest effective examples is the grid world example ([15] and Section 4.2.5 of [14]), which uses a two-dimensional, often 10×10 world, where states are locations in the grid, and actions are movements up, down, left, or right within the grid. A transition function specifies stochastic transitions between locations, and a reward function dictates that certain rewards are given at certain locations.

Building off this base, much more complex scenarios can be imagined, and in fact have been implemented. As an example, using more complex world, transition, and reward models, we can model numerous complex problems, including autonomous robot/vehicle navigation, exploration, and interaction with objects (grasping) and humans (following) [12, 13, 16], along with aircraft collision avoidance (explored consistently in [14], and exclusively and in detail in chapter 10). In these cases, we even add further complication to the problem by making the agent unable to (fully) observe its state:the agent cannot fully observe its state, and must instead maintain a belief over its possible states at any given time; this is known as a partially-observable MDP (POMDP; [12] and others) when the agent cannot directly observe its state at all, or a mixed-observability MDP (MOMDP; [16]) when the agent can directly observe certain variables in its state but not all of them. For example, both of the settings mentioned above have been shown to have effective POMDP/MOMDP formulations that can be solved to find approximately-optimal policies using techniques such as SARSOP [12], as value iteration does not generalize well to the POMDP setting.

2.2 Differences from current work

It is important to note that, at least in this small subset of examples known to the author, many mainly focus on finding policies that will allow an agent to do one fixed task that, more or less, has a single "phase":

- In the case of the grid world example [15] and the underwater navigation [12, 16] problem, the agent must navigate once from a starting point to an optimal terminal/destination state.
- In the case of the person-following/assistance problem [12], the robot has only one target, although the location of that target does change.
- In the case of the grasping example [12, 13], the agent must perform manuevers to grasp a single target object using robotic fingers and tactile/contact sensors.
- In the case of a simplified aircraft collision avoidance setting, in the typical case, there are two vehicles, and the system must decide on optimal action(s) to take to ensure avoidance of a collision; however, each time this occurs, while the locations and velocities of the aircraft vary, the setting is relatively constant otherwise, and the objective is identical.

In the current work, we present a system that has multiple distinct "phases", based on the number of habitat components that have been placed at their optimal locations at any given time. We do this without formulating explicitly-different MDPs for each phase, and instead build this into the reward/transition information. This is minimally coded into the robot's state, allowing us to constrain the state space as much as possible. Within each of these phases, there are also two "subphases" associated - obtaining a new component from the base location, and placing this new component at the target location. See Section 3.1 below for details. However, due to the simplicity of the current work's uncertainty assumptions, setting, and simulation, it is important to note the greatly-increased complexity in many of the problems above (especially the POMDP problems); the distinction regarding "phases" is relatively minor and semantic in comparison.

3 Technical details

3.1 Problem overview

In this problem, we adopt the setting of a two-dimensional, grid-world-like world, with discrete grid locations for an agent to move between. In any given world, there are a few different types of objects:

- The robot agent itself (R), which will be moving throughout the world and interacting with habitat component objects.
- The habitat component objects, representing pieces of the habitat that must be placed in its desired location. Each habitat component, h, has a specific location it must be placed at, L_h. A strict order of placement of components is enforced, and the reward function implicitly communicates L_h and this order to the robot (see Section 3.3). When describing order, we may explicitly refer to each component as h_i, for some $i \in \{1, \ldots, |H|\}$, where H is the set of all habitat components, and $|H|$ is the total number of habitat components for the current world. Appropriately, the corresponding target location for habitat component h_i is explicitly written L_{h_i}.

- The base (B) at a specific fixed location (L_B), where all habitat components $h \in H$ initially are located. For each $h \in H$, the robot must pick up h from B, move h to its target location L_h, and place h at that location. It must return to the base to pick up any subsequent components before placing these later components at their destinations.
- Obstacles are scattered throughout each world. Each obstacle (o in the full set of obstacles O of size $|O|$) may cause harm to the robot's physical parts, such as damaging a wheel, actuator, gear, etc., or in the worst case even toppling the robot (which could be especially disastrous if the robot is carrying a habitat component at the time). While these specific outcomes are not explicitly modeled or simulated, see Section 3.6 for how this is implicitly incorporated into our formulation.

Thus, we note that the "phases" mentioned in Section 2.2 refer to these cycles of having to pick up, move, and place habitat components during the robot's operation in each world for each habitat component. The first "subphase" includes navigating to the base B at location L_B, picking up a single habitat component h_i from B, moving to the desired location L_{h_i}, and placing the habitat component down at that location. The second subphase involves navigating back to base B from location L_{h_i} in order to start the next cycle for $L_{h_{i+1}}$.

3.2 Worlds

We use three distinct worlds, as shown in Figures 2, 3, and 4. For simplicity, we will refer to $(1, 1)$ as the bottom-left-most location in any world, with the x/first coordinate increasing to the right, and the y/second coordinate increasing upwards.

The "small" world (Figure 2) is a 5×5 world, with 4 total habitat components ($|H| = 4$), and 3 total obstacle components ($|O| = 3$). The base B is located at $L_B = (5, 5)$ (upper-right-most corner). The "medium" world (Figure 3) is a 10×10 world, with 6 total habitat components ($|H| = 6$), and 6 total obstacle components ($|O| = 6$). The base B is located at $L_B = (1, 1)$ (lower-left-most corner). The "large" world (Figure 4) is a 20×10 world, with 15 total habitat components ($|H| = 15$), and 6 total obstacle components ($|O| = 6$). The base B is located at $L_B = (20, 10)$ (center-right boundary).

3.3 States

We will be modeling this problem as an MDP (see Section 8 for explanations and future directions); thus, all variables in the state are fully observed by the robot agent at all times. At any given point in time, the robot is in a state $s \in S$ (where S is the full set of all possible states the robot can take in a given world) of the following form:

$$s_t = \big(x_t, y_t, d_t, p_t, c_t\big) \tag{3}$$

Where:

- x_t is an integer representing the x location of the robot at time t (left-right location in the world). This is initially set to the x component of the base B's location L_B for the given world.
- y_t is an integer representing the y location of the robot at time t (down-up location in the world). This is initially set to the y component of the base B's location L_B for the given world.
- d_t is one of $\{d_{\text{up}}, d_{\text{down}}, d_{\text{left}}, d_{\text{right}}\}$, indicating the direction the robot is facing at time t (up, down, left, right). This is initially set to d_{up}.
- p_t is an integer between 0 and $|H|$ representing the number of habitat components that have not yet been placed at their target locations. Note that this still includes a habitat component even if the robot is currently carrying it to its location, but has not yet actually placed it at that location. As such, this is initialized to $|H|$. Use of the letter p corresponds to the property of this state variable in determining what $p$hase the robot is currently in at time t. See Sections 3.5 and 3.6 for more details on how this works in practice.
- c_t is a boolean value (true or false) representing if the robot is currently carrying a habitat component at time t. This is initially set to false, as the robot starts with no habitat components being carried.

No other information is explicitly given to the robot. In fact, to keep the state space minimal and appropriately constrained, this is not encoded anywhere explicitly into the state, and the robot cannot access this as a world/global variable anywhere. Instead, this is modeled in the transitions and rewards functions (again, see Sections 3.5 and 3.6). The robot essentially follows the rewards as derived in an optimal policy solver (Section 3.7).

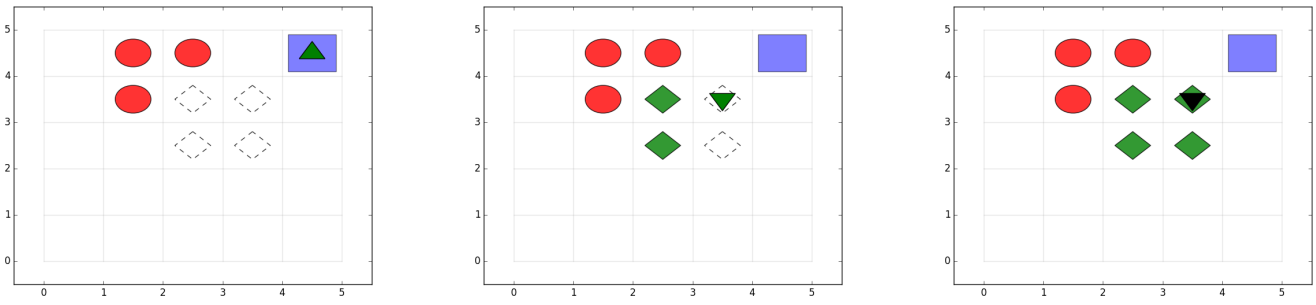
As a final note, we mention that the robot's location can actually move 1 step "out of bounds" in either the x or y direction. The robot can do this intentionally if it wants, but more often than not will only enter these states when it stochastically enters such a state accidentally when moving forward near a boundary. This is penalized in the reward function.

For the small world, there are 1960 states (7 possible x locations including out-of-bounds on either end, 7 possible y locations also including out-of-bounds, 4 possible directions the robot is facing, 5 possible values for $p_t \in [0, 5]$, and 2 possible boolean values for c_t). Generalizing the same countings of variables, the medium world has 8064 possible states, and the large world has 47872 possible states.

3.4 Actions

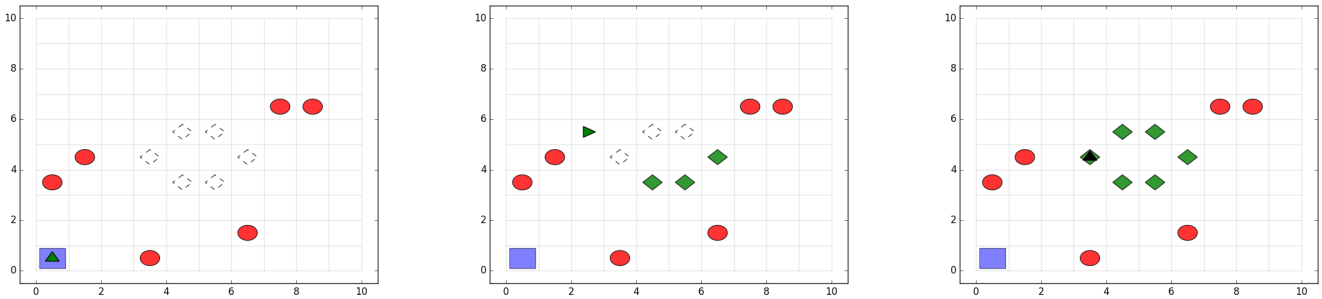
Actions are significantly simpler. There are only 5 different actions that the robot can execute, and the robot is able to execute them at any time t and thus at any state $s \in S$:

- Move forward. The robot attempts to move forward one step. Probabilistically (according to the transition function), the robot ends up in the forward location some fraction of the time, but otherwise can end up in the forward-left, forward-right, or same/unmoved location.
- Turn left. The robot attempts to turn left to face a new direction. Similarly, the robot ends up turning some fraction of the time, but otherwise ends up not turning at all (facing the same direction it started in).
- Turn right. Similar to the left-turn action.
- Pick up a habitat component. The robot attempts to pick up a habitat component. If at the base B, this succeeds with some probability, and otherwise fails.



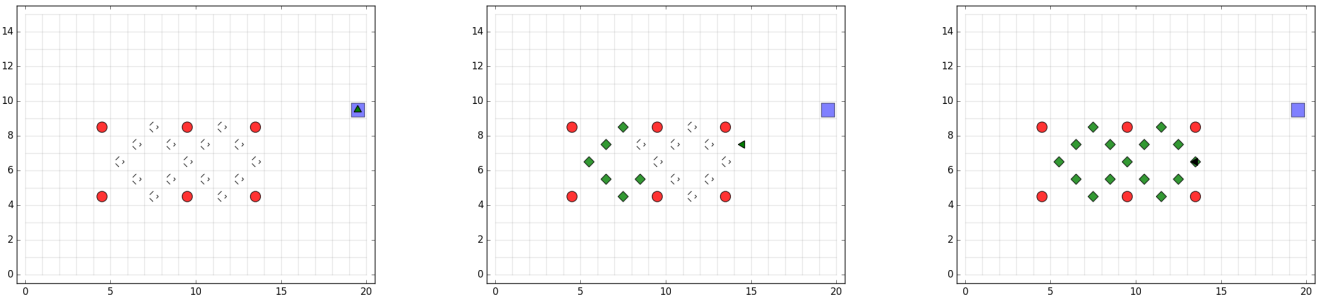
(a) Initial scenario, small world. **(b)** Intermediate scenario, small world. **(c)** Ending scenario, small world.

Figure 2: Examples of progress through small world simulation. See Section 3.9 for details on shape/color meanings.



(a) Initial scenario, medium world. **(b)** Intermediate scenario, medium world. **(c)** Ending scenario, medium world.

Figure 3: Examples of progress through medium world simulation. See Section 3.9 for details on shape/color meanings.



(a) Initial scenario, large world. **(b)** Intermediate scenario, large world. **(c)** Ending scenario, large world.

Figure 4: Examples of progress through large world simulation. See Section 3.9 for details on shape/color meanings.

- Place a habitat component. The robot attempts to place a habitat component at its current location. If at the proper location and carrying the proper component, this succeeds with some probability, and otherwise fails (the component remains in the robots possession).

3.5 Transitions

We now describe our transition function $T(s' \mid s, a)$ (or equivalently, $T(s_{t+1} \mid s_t, a_t)$).Given the robot is in state s_t and executes action a_t, we determine the new/next state s_{t+1} as follows:

- If the robot is currently out-of-bounds in either the x or y direction, then regardless of action, the robot deterministically moves to the nearest in-bounds location in the next time step.
- If the robot is in a terminal state ($p_t = 0$, since the robot has placed all habitat components at their proper locations), then the robot deterministically transitions to the same terminal state at the next time step, and stays there forever.
- If the robot tries to move forward, with probability p_{movement}, the robot ends up in that expected forward/"success" location. Otherwise, the robot ends up in the forward-left, forward-right, or same/unmoved location with probability $\frac{(1 - p_{\text{movement}})}{3}$ for each of the three "failure" locations. Overall, state variables (x_t, y_t) are updated to (x_{t+1}, y_{t+1}) accordingly, while all other state variables remain unchanged.
- Similarly, if the robot tries to turn left or right, with probability p_{movement}, this succeeds, and the robot ends up facing the desired direction in the next time step. However, with probability $(1 - p_{\text{movement}})$, the robot remains facing the same direction in the next time step as it is in the current time step. Overall, state variable d_t is updated to d_{t+1} accordingly.
- If the robot attempts to pick up a habitat component, is at the world's base location L_B, is not already carrying a component, *and* there is at least one component still remaining in the base ($p_t > 0$), *then* with probability $p_{\text{interaction}}$, the robot transitions into a new/next state where it is carrying a component. Otherwise, if these criteria are not met, or if they are but with probability $(1 - p_{\text{interaction}})$, the robot remains in the same state it is currently in. Overall, state variable c_t is updated to c_{t+1} accordingly.
- If the robot attempts to place a habitat component, is currently carrying a component, there is at least one component un-placed ($p_t > 0$), *and* is at the proper current target location, *then* with probability $p_{\text{interaction}}$, the robot successfully places the component and transitions into a new state where there is one fewer component remaining (p_t is decremented by 1). Otherwise, if these criteria are not met, or if they are but with probability $(1 - p_{\text{interaction}})$, the robot remains in the same state it is currently in. Overall, state variable p_t is updated to p_{t+1} accordingly.

To determine if the robot is at the "proper current location" for placing a component, we use its current state's p_t value, which we recall is an integer that ranges from 0 to $|H|$ (the number of total habitat components we have to place in the current world). We also note that the world "knows" its own target habitat component locations (L_{h_i} for each $h_i \in H$); this means the transition function can access (i.e., loop through and examine) each of the target habitat component locations and their corresponding order. Thus, if $p_t = |H|$, no components have been placed, and the transition function looks at the first component (h_1) that should be placed to find its target location (L_{h_1}); if the robot's current location (x_t, y_t) is equal to the first target location L_{h_1}, the robot can successfully place the component at its current location (i.e., it is at its "proper current target location"). Similarly, if $p_t = |H| - 1$, then the robot must be at location L_{h_2}.[1]

Lastly, we note that we can set the probabilities p_{movement} and $p_{\text{interaction}}$ accordingly for each run, as will see in Section 4.

3.6 Rewards

Our reward function, $R(s_t, a_t)$, has conditional cases as follows:

- If the robot is in a terminal state, it receives 0 reward forever. This is to avoid infinite reward when solving for optimal policies.
- The robot receives a small penalty $\mathfrak{p}_{\text{movement}}$ with every attempted forward movement action, and a small penalty $\mathfrak{p}_{\text{turn}}$ with every attempted turn-left or turn-right action.
- The robot receives a penalty $\mathfrak{p}_{\text{out-of-bounds}}$ when attempting to move out of bounds.
- If the robot is currently at the base location L_B, is not carrying a habitat component already, and there is at least one habitat component remaining to be placed ($p_t > 0$), and the robot executes the pick-up action, then it receives reward $\mathfrak{r}_{\text{pickup}}$ for a valid pick-up of a habitat component from the base. Otherwise, if the robot is not at a base location and tries to pick up a component, it receives penalty $\mathfrak{p}_{\text{pickup,not-base}}$; if already carrying a component and trying to pick up another, it receives penalty $\mathfrak{p}_{\text{pickup,carrying}}$.
- If the robot currently should be placing h_i according to its state variable p_t (see Section 3.5), and is at location L_{h_i}, and the robot executes the place-component action, then it receives reward $\mathfrak{r}_{\text{place}}$ for placing a habitat component at its target location. Otherwise, if it tries to place a component at the wrong location for that specific component (carrying h_i and tries to place at some location that is not L_{h_i}), then it receives penalty $\mathfrak{p}_{\text{place,wrong}}$. If it is not holding a component and tries to place one, it receives penalty $\mathfrak{p}_{\text{place,invalid}}$.
- If the robot attempts to *intentionally* move forward

[1] In hindsight, it would have been simpler to simply keep an *increasing* instead of *decreasing* counter here, such that our current state encodes how many components we *have already placed* instead of the number remaining. Though perhaps this is a trivial point, as this only saves one trivial algebraic computation.

to the location of a habitat component that it has already placed (reward function can check this by using p_t and the locations and orders of all the habitat components) or is standing on this location and attempting to turn instead of moving away, then the robot receives a penalty $\mathfrak{p}_{\text{previous-target}}$ Note that we intentionally want it to be un-penalized to step on the current habitat component's target location to place it at that location without penalty, and do not penalize for walking over locations that have not been used yet.

- If the robot attempts to *intentionally* move forward to the location of an obstacle ($o \in O$) or ends up on an obstacle location stochastically, it receives a penalty $\mathfrak{p}_{\text{obstacle}}$.

For reference, we used values for rewards and penalties given in Table 1. These were fine-tuned to give the desired behavior. See Section 9.

3.7 Solving

Given the worlds, states, actions, transition function, and reward function defined above, we use value iteration to solve for optimal policies. The POMDPs.jl and DiscreteValueIteration.jl Julia packages were used (see Section 3.8) to perform value iteration given the manually-coded models and functions by the author to interface with this package.

3.8 Implementation

The Julia language [5] was used for all code associated with this project. The POMDPs.jl library [9], associated DiscreteValueIteration.jl library, and all other associated packages were used for aiding in formulation and solving of this MDP problem as described above. PyPlot.jl [10] was used for visualization (Section 3.9).

3.9 Visualization

PyPlot.jl for Julia was used for the simulation visualizations. In Figures 2, 3, and 4, along with the videos mentioned in Section 4, we use the following scheme for colors and shapes: the triangle is the robot agent and also indicates direction it is facing; t

- The triangle represents the robot agent's current location and direction it is facing. The robot is black if not carrying a habitat component, and green if it is carrying one. In the videos, the robot flashes red if it ends up in a state that it did not explicitly attempt to move into (see Section 3.5).
- The base is represented with a blue-purple square, and does not change.
- Habitat component target locations are represented by green diamonds. Before the appropriate component is placed at a target location, the diamond is not filled, and has a dotted-line border; after the appropriate component is placed, the diamond becomes filled.
- Obstacles are represented by red circles, and do not change.

Faint gray grid lines indicate the gridded locations in the world.

4 Results

Value iteration provided an estimate of the optimal policy for each case. Value iteration was run to convergence each time (always converged in 111 iterations), with a Bellman residual of 1e-3. To aid in evaluation, we simulated runs in each world and with varying parameters. The robot began at the base location each time, and proceeded to follow the policy it derived through value iteration. Rewards were tracked at each step. A video of the simulations running is linked at [1]. Representative images showing progress throughout a simulation are shown in Figures 2, 3, and 4.

Tables 2, 3, and 4 show performance metrics for each of the three worlds. For each row in each table, value iteration and simulation was performed 3-4 separate times (with different random number generator seeds), and thus ranges are reported where applicable. Run-time did not differ significantly across conditions within each world; value iteration always took between 1.8-1.9 seconds on the small world, 10.5-11.0 seconds on the medium world, and 126.0-132.0 seconds on the large world. Rows in each table correspond to different settings of the primary uncertainty/stochastic-behavior-determining probabilities, p_{movement} and $p_{\text{interaction}}$, which essentially determine how sure we can be that we will end up in a state we intend to transition to (see Section 3.5).

In each table, "total simulated reward" refers to the range of rewards accrued over the distinct simulations for that entry/case. "Benchmark rewards" refer to the total rewards the author could manually achieve using manually-chosen optimal actions at each relevant state; equivalently, since all policies under no stochastic behavior produced equivalently-optimal policies (0 sub-optimal moves), the benchmark could also be seen as the reward accrued under no stochastic behavior. This occurs when both p_{movement} and $p_{\text{interaction}}$ are set to 1.0, meaning the robot always moves, turns, picks up components, and places components exactly as it intends to. Note that this has one fixed value for each world, and does not change with probability conditions. As a result, "simulated vs benchmark reward" is the range of percents of the single benchmark reward that we achieve in the simulations for that case, and is computed directly from the "total simulated reward" columns.

5 Conclusion

6 Analysis

Overall, the formulations described above produce excellent results in the simplistic worlds we simulate here. In each world, the robot never gets "stuck" in any states (such as spinning around or walking back and forth), and instead always succeeds in delivering all habitat components to their proper locations. Even under relatively generous uncertainty/stochastic-behavior in transitions (the $p_{\text{movement}} = p_{\text{interaction}} = 0.7$ cases in the result tables), the agent still performs extremely well. Qualitatively, we can see in the video at [1] that the system seems to perform optimally in every case, and overall seems to take actions exactly as a human operator would command; we explicitly compare this for the no-uncertainty cases with the benchmarks in Section

Table 1: Reward and penalty values

$\mathfrak{r}_{\text{pickup}}$	100
$\mathfrak{r}_{\text{place}}$	1000
$\mathfrak{p}_{\text{movement}}$	-1
$\mathfrak{p}_{\text{turn}}$	-1
$\mathfrak{p}_{\text{out-of-bounds}}$	-50
$\mathfrak{p}_{\text{pickup,not-base}}$	-30
$\mathfrak{p}_{\text{pickup,carrying}}$	-100
$\mathfrak{p}_{\text{place,wrong}}$	-100
$\mathfrak{p}_{\text{place,invalid}}$	-50
$\mathfrak{p}_{\text{previous-target}}$	-100
$\mathfrak{p}_{\text{obstacle}}$	-100

Table 2: Rewards achieved using value iteration for simulations in the small world.

p_{movement}	$p_{\text{interaction}}$	Total simulated reward	Benchmark reward	Simulated vs benchmark reward (%)
1.0	1.0	4360	4360	100
0.9	0.9	4458 - 5362	4360	102.2 - 123.0
0.9	1.0	4355 - 4359	4360	99.9
0.7	0.7	5035 - 6648	4360	115.5 - 152.5
0.7	1.0	3945 - 4348	4360	90.5 - 99.7

4, but future work would also include additional benchmarks (see Section 8).

However, this latter remark brings up an important point and observation: the agent actually accrues *more* reward as specific certainties decrease, uncertainties increase, and stochastic behavior becomes more common. In particular, in each of the three cases, we see we actually accrue the *most* reward when we are the least certain ($p_{\text{movement}} = p_{\text{interaction}} = 0.7$)! This seems non-obvious at first, but becomes clear when we realize that these probabilities not only determine our transition function but also our simulation/world behavior. In particular, we see from Section 3.6 and Table 1 that we receive large positive rewards when simply *attempting to* pick up habitat components from the base or place them at their appropriate locations. In particular, the lower $p_{\text{interaction}}$ is (the variable that controls how likely we are to succeed when picking up or placing a component), the more likely it is we will fail when we attempt to pick up or place a component - however, we still receive our reward due to the intention of our action and our definition of our reward function! As a result, we indeed often receive 2-3 positive rewards consecutively in the $p_{\text{interaction}} = 0.7$ cases, as we fail to execute a pick-up/place action the first one or two times, and then succeed, gathering large rewards each time regardless.

To account for this, we included the cases where $p_{\text{interaction}} = 1.0$, while still decreasing p_{movement}. This means we deterministically succeed when interacting with habitat components (picking up and placing), but still often will move and turn to new states we did not intentionally mean to transition to. We see that in cases where $p_{\text{interaction}} = 1.0$ and $p_{\text{movement}} < 1.0$, we actually have decreased rewards compared to our benchmarks for each world, and never achieve the benchmark. This is usually because the unintended movements induced by $p_{\text{movement}} < 1.0$ cause us to either fail to turn when we want to, or move forward to unintended locations; making up for these errors causes us to lose a small amount of reward due to the -1 penalties on movement and turning we described in our reward function.

It is also worth noting that we could have also penalized for simply ending up in states where we are standing on past target locations even when we did not intend to move to that location (due to stochastic behavior, we ended up at a past target location). In practice, penalizing states on past target locations led to some non-obvious issues - in particular, in the large world, after placing the center-most habitat component, when facing left, the robot would simply move forward into the left, already-filled array of components, and would get stuck there indefinitely. It would avoid simply turning around while on the component it just placed because it would be accruing some penalties, and would instead move off the past target location as soon as possible. More work could be done in the future to mitigate this and produce better policies that would be able to escape/avoid such a situation, even if having to accrue penalties by turning on top of a past target location. A very small penalty could also have been used, instead of a larger one; more testing would be necessary to find an appropriate penalty.

7 Complications

This project gave the author valuable insight into formulating MDPs, especially with regards to the actual choosing of rewards and penalties in a balanced manner to facilitate intended behavior overall. Especially in initial tests where "optimal" rewards and penalties have not been chosen, the performance of the agent and value iteration can vary wildly, with very unfavorable results that make it hard to pin down if these poor results are due to bugs or poor reward/penalty choices (or both). Some unexpected and unfavorable results can occur with even slight imbalances between rewards and penalties; for example, as mentioned above, in more complex situations such as the center-most habitat component in the large world, non-trivial penalties on staying on past target

Table 3: Rewards achieved using value iteration for simulations in the medium world.

p_{movement}	$p_{\text{interaction}}$	Total simulated reward	Benchmark reward	Simulated vs benchmark reward (%)
1.0	1.0	6469	6469	100
0.9	0.9	6461 - 8362	6469	99.9 - 129.3
0.9	1.0	6358 - 6465	6469	98.3 - 99.9
0.7	0.7	8534 - 9036	6469	131.9 - 139.7
0.7	1.0	6042 - 6120	6469	93.4 - 94.6

Table 4: Rewards achieved using value iteration for simulations in the large world.

p_{movement}	$p_{\text{interaction}}$	Total simulated reward	Benchmark reward	Simulated vs benchmark reward (%)
1.0	1.0	16054	16054	100
0.9	0.9	16808 - 19008	16054	104.7 - 118.4
0.9	1.0	15526 - 15823	16054	96.7 - 98.6
0.7	0.7	20746 - 27111	16054	129.2 - 168.9
0.7	1.0	15311 - 15578	16054	95.4 - 97.0

locations (which seems like a harmless thing) would cause the robot to enter a situation it could not get out of given its policy, and it would simply spin around inside the left array of target locations forever. Again, work could be done to come up with policies that would prevent this, but it was still a valuable lesson to see the small effects changes in rewards and penalties can have in overall situation outcome.

8 Future work

In addition to the improvements addressed in the previous two sections, many more steps could be taken to further improve this problem formulation and the associated approaches. In particular:

- Other solvers. While the author did some experimentation with using Monte Carlo Tree Search (MCTS), not enough headway was made on that front to warrant inclusion into this report.
- Add new "benchmark" under uncertainty, where we quantify the maximum/optimal rewards achievable by the optimal policy and/or a manual human operator under uncertainty conditions (specifically, where $p_{\text{movement}} < 1.0$ and $p_{\text{interaction}} = 1.0$ for maximal usefulness of our benchmark). This would allow us to directly compare the performance of computed policies under uncertainty directly to an optimal benchmark/manual policy obtained by a human under uncertainty conditions. Making this direct comparison would allow us to quantify exactly how much value/information is added or lost by formulating this problem as an MDP. *Informally*, it *seems* the system described above performs optimally and equivalently to such a benchmark under uncertainty conditions (see the linked videos of the system in action); however, this was not explicitly quantified, and it would be useful to do so.
- POMDPs. The problem could easily be extended to a POMDP/MOMDP. For example, in reality, the robot would not precisely know its exact (x, y) location, and would instead use an array of onboard sensors to try to locate itself within a given geographical zone. The robot might have some nearby landmarks to calibrate its location from using computer vision, or might have some method of calibration with regards to a known, ground-truth location of the base B. The robot would then maintain a belief over its possible location states (partially-observable x and y estimations); recalibration actions would take some time and have some cost/penalty, but could provide more accurate and pinpointed beliefs; the robot would need to be quite certain of its exact location before placing a habitat component, and would therefore need to recalibrate every few steps. Solvers such as SARSOP would likely perform quite well in this situation (while QMDP might not work quite as effectively as SARSOP since some recalibration/information-gathering actions would be necessary from time to time).
- More realism. In practice, a real simulation would have complex models of the robot's movements, functionality, power levels, sensors, etc. More of these variables could be added to the robot's state, interaction with objects, and interaction with its world.

9 Summary

In conclusion, this project proved successful in formulating and simulating a simple MDP where a Mars "hab-bot" must obtain habitat components from a base, move them to specific locations, and place them at these locations in a specific order. We kept the state space relatively minimal and manageable (at least for these example problems), and instead modeled most of the complexities of the scenario implicitly into the transition and reward functions. Our robot agent never explicitly accesses information about target or obstacle locations, but instead uses value iteration to come up with optimal policies based on the effects these target and obstacle locations have on the transition and reward functions. In doing so, we can efficiently encode different, evolving, complex "phases" of the task with minimal state dimensionality.

While the simplicity of this problem formulation hinders this project from making contributions at the level of serious, bleeding-edge work in the field of space-based robot navigation and planning, it certainly did give the author great hands-on experience with formulating, fine-tuning, and solv-

ing non-trivial MDP problems. It also allowed to accessibly work (at least to some extent) in the problem domain of designing autonomous robots for building human-habitable constructs on Mars, which is otherwise quite an inaccessible field. Because of all this, overall the author considers this project a big success, and enjoyed exploring novel applications of MDPs to complex problems, especially in this problem setting.

References

[1] Abraham Botros - Personal Website - Mars Hab Bot. http://www.abrahambotros.com/#MarsHabBot, and https://www.youtube.com/watch?v=C-oJjzNpFag.

[2] Communications with earth: Data rates/returns. http://mars.jpl.nasa.gov/msl/mission/communicationwithearth/data/.

[3] Communications with earth: How the rovers can communicate through mars-orbiting spacecraft. http://mars.nasa.gov/mer/mission/comm_orbiters.html.

[4] Elon musk wants to build 80,000-person mars colony. http://www.wired.com/2012/11/elon-musk-mars-colony/.

[5] Julia. http://julialang.org/.

[6] Mars analog habitat. https://en.wikipedia.org/wiki/Mars_analog_habitat.

[7] Nasa picks winners for 3d-printed mars habitat design contest. http://www.space.com/30854-nasa-3d-printed-mars-habitat-contest-winners.html.

[8] Nasa releases plan outlining next steps in the journey to mars. http://www.nasa.gov/press-release/nasa-releases-plan-outlining-next-steps-in-the-journey-to-mars.

[9] Pomdps.jl: Api for solving pomdps in julia. https://github.com/sisl/POMDPs.jl.

[10] Pyplot.jl: The pyplot module for julia. https://github.com/stevengj/PyPlot.jl.

[11] Richard Bellman. *Dynamic Programming*. Princeton University Press, Pinrceton, NJ, 1 edition, 1957.

[12] Wee Sun Lee Hanna Kurniawati, David Hsu. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008.

[13] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Grasping POMDPs. In *IEEE International Conference on Robotics and Automation*, 2007.

[14] Mykel J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[15] Poole David L. and Mackworth Alan K. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, New York, NY, USA, 2010.

[16] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee. POMDPs for robotic tasks with mixed observability. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.